LAB 6 REPORT

Course: CPS633

Section: 6

Group Members:

Alishba Aamir, 500974648

Ivan Golovine, 500813431

Fangbo Ren, 500884730

Task 1: Deriving the Private Key

For task one, in our rsa.c file, we performed the caculation p*q and let (e,n)in our main file, we assigned the three prime numbers p,q and e. As explained in the rsa slides, we multipled p-1 and q-1. The computed the private key res by calculating the mod inverse (ed mod φ(n) = 1) .

This gets us:

Public key : (e,n)

Private key = res

```
Terminal
#include "rsa.c"
#include "print_convert.c"

int main (){


        BIGNUM *p = BN_new();
        BIGNUM *q = BN_new();
        BIGNUM *e = BN_new();

        // p
        BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");

        // q
        BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");

        // e
        BN_hex2bn(&e, "0D88C3");

        BIGNUM* p_key = get_private_key(p, q, e);
        printBN("The private key is:", p_key);

        BIGNUM* enc = BN_new();
        BIGNUM* dec = BN_new();
}
~
```

```
Terminal
#include "rsa.h"

BIGNUM* get_private_key(BIGNUM* p, BIGNUM* q, BIGNUM* e)
{

        BN_CTX *ctx = BN_CTX_new();
        BIGNUM* p_minus_one = BN_new();
        BIGNUM* q_minus_one = BN_new();
        BIGNUM* one = BN_new();
        BIGNUM* tt = BN_new();

        BN_dec2bn(&one, "1");
        BN_sub(p_minus_one, p, one);
        BN_sub(q_minus_one, q, one);
        BN_mul(tt, p_minus_one, q_minus_one, ctx);

        BIGNUM* res = BN_new();
        BN_mod_inverse(res, e, tt, ctx);
        BN_CTX_free(ctx);
        return res;
}
```

```
Terminal
[11/07/21]seed@VM:~/.../lab7$ gcc task1.c -lcrypto
[11/07/21]seed@VM:~/.../lab7$ ./a.out
The private key is: 0x3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B
496AEB
[11/07/21]seed@VM:~/.../lab7$
```

Task 2: Encrypting a Message

In this task we first find the hex value of our message that we want to encrypt.

```
[11/07/21]seed@VM:~/.../lab7$ python -c 'print("A top secret!".encode("hex"))'
4120746f70207365637265742l
[11/07/21]seed@VM:~/.../lab7$
```

In our main file, we assigned variables for the private key, public key, and e value which is the modulus

```
// private key
BIGNUM* private_key = BN_new();
BN_hex2bn(&private_key, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

//public key
BIGNUM* public_key = BN_new();
BN_hex2bn(&public_key, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
printBN("Public key: ", public_key);

// e
BIGNUM* mod = BN_new();
BN_hex2bn(&mod, "010001");


BIGNUM* message = BN_new();
BN_hex2bn(&message, "4120746f702073656372657421");

printBN("Plaintext message: ", message);
enc = encrypt_message(message, mod, public_key);
printBN("Encrypted message", enc);
dec = decrypt_message(enc, private_key, public_key);
printf("Decrypted message: ");
printHX(BN_bn2hex(dec));
printf("\n");
```

In our rsa.c file, we use the formula message^mod (modulo public_key) to encrypt the message

```
BIGNUM* encrypt_message(BIGNUM* message, BIGNUM* mod, BIGNUM* public_key){
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM* enc = BN_new();
        BN_mod_exp(enc, message, mod, public_key, ctx);
        BN_CTX_free(ctx);
        return enc;
}
```

This is the encrypted Message

```
Terminal
[11/07/21]seed@VM:~/.../lab7$ gcc lab7.c -lcrypto
[11/07/21]seed@VM:~/.../lab7$ ./a,out
bash: ./a,out: No such file or directory
[11/07/21]seed@VM:~/.../lab7$ ./a.out
Private key: 0x182363E2DA763AD4DC94DBE64CD6869FEDD1B10B1E8810416A9CD4E9AF6B7FC5
Public key:  0xDCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5
Plaintext message:  0x4120746F702073656372657421
Encrypted message 0x6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
Decrypted message: A top secret!

[11/07/21]seed@VM:~/.../lab7$
```

Task 3: Decrypting a Message

To decrypt the message, we assign the hex version of the encrypted message to a varible

```
BIGNUM* encrypted_message = BN_new();
BN_hex2bn(&encrypted_message, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");

dec = decrypt_message(encrypted_message, private_key, public_key);
printf("Decrypted message, TASK3: ");
printHX(BN_bn2hex(dec));
```

In our rsa.c file , we use the formula (message ^ mod) ^ public_key to decrypt the message

```
BIGNUM* decrypt_message(BIGNUM* enc, BIGNUM* private_key, BIGNUM* public_key)
{

        BN_CTX *ctx = BN_CTX_new();
        BIGNUM* dec = BN_new();
        BN_mod_exp(dec, enc, private_key, public_key, ctx);
        BN_CTX_free(ctx);
        return dec;
}
```

```
Decrypted message, TASK3: Password is dees
```

Task 4: Signing a Message

```
[11/08/21]seed@VM:~/.../lab7$ python -c 'print("I owe you $2000".encode("hex"))'
49206f776520796f75202432303030
[11/08/21]seed@VM:~/.../lab7$ python -c 'print("I owe you $3000".encode("hex"))'
49206f776520796f75202433303030
```

The messages differ very slightly when converted to hexcode.

```
BN_hex2bn(&m, "49206f776520796f75202433303030");
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

BN_hex2bn(&e, "010001");
```

The above message is the 2000$ one.

```
BN_hex2bn(&m, "49206f776520796f75202432303030");
BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

BN_hex2bn(&e, "010001");
```

The above message is the 3000$ one.

```
[11/08/21]seed@VM:~/.../lab7$ gcc bn_sample.c -lcrypto
[11/08/21]seed@VM:~/.../lab7$ ./a.out
s =  80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82
s =  04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2EB
[11/08/21]seed@VM:~/.../lab7$
```

However, after signing the message by applying the RSA function to it with a private key they differ significantly.

Task 5: Verifying a Signature

First we got the hex version of the message to compare.

```
[11/08/21]seed@VM:~/.../lab7$ python -c 'print("Launch a missile.".encode("hex"))'
4c61756e63682061206d697373696c652e
```

We then input the message, Signature and Alice's public key. We also created 2 versions of the signature to test out if the verification works.

```
    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115
");
    BN_dec2bn(&e, "65537");
    BN_hex2bn(&M, "4c61756e63682061206d697373696c652e");
    BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F
");
    //BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB680
3F");
```

First we tested the valid signature and after comparing the message with the by doing BN_mod_exp(C,S,e,n,ctx). We got a valid signature.

```
[11/08/21]seed@VM:~/.../lab7$ gcc bn_sample.c -lcrypto
[11/08/21]seed@VM:~/.../lab7$ ./a.out
Valid Signature!
```

Next we tested the modified signature and it returned invalid even with a single modification.

```
[11/08/21]seed@VM:~/.../lab7$ gcc bn_sample.c -lcrypto
[11/08/21]seed@VM:~/.../lab7$ ./a.out
Invalid
```

Task 6: Manually Verifying an X.509 Certificate

Step 1: Download a certificate from a real web server.

We used the seedsecuritylabs website to get our certificates for testing and saved them in c0.pem and c1.pem.

```
[11/08/21]seed@VM:~/.../lab7$ openssl s_client -connect seedsecuritylabs.org:443 -s
howcerts > task6.txt
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assuran
ce EV Root CA
verify return:1
depth=1 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High As
surance Server CA
verify return:1
depth=0 C = US, ST = California, L = San Francisco, O = "GitHub, Inc.", CN = www.gi
thub.com
verify return:1
```

```
r CA
   i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert High Assurance
-----BEGIN CERTIFICATE-----
```

```
MIIEsTCCA5mgAwIBAgIQBOHnpNxc8vNtwCtCuF0VnzANBgkqhkiG9w0BAQsFADBs
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
d3cuZGlnaWNlcnQuY29tMSswKQYDVQQDEyJEaWdpQ2VydCBIaWdoIEFzc3VyYW5j
ZSBFViBSb290IENBMB4XDTEzMTAyMjEyMDAwMFoXDTI4MTAyMjEyMDAwMFowcDEL
MAkGA1UEBhMCVVMxFTATBgNVBAoTDERpZ2lDZXJ0IEluYzEZMBcGA1UECxMQd3d3
LmRpZ2ljZXJ0LmNvbTEvMC0GA1UEAxMmRGlnaUNlcnQgU0hBMiBIaWdoIEFzc3Vy
YW5jZSBTZXJ2ZXIgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC2
4C/CJAbIbQRf1+8KZAayfSImZRauQkCbztyfn3YHPsMwVYcZuU+UDlqUH1VWtMIC
Kq/QmO4LQNfE0DtyyBSe75CxEamu0si4QzrZCwvV1ZX1QK/IHe1NnF9Xt4ZQaJn1
itrSxwUfqJfJ3KSxgoQtxq2lnMcZgqaFD15EWCo3j/018QsIJzJa9buLnqS9UdAn
4t07QjOjBSjEuyjMmqwrIw14xnvmXnG3Sj4I+4G3FhahnSMSTeXXkgisdaScus0X
sh5ENWV/UyU50RwKmmMbGZJ0aAo3wsJSSMs5WqK24V3B3aAguCGikyZvFEohQcft
bZvySC/zA/WiaJJTL17jAgMBAAGjggFJMIIBRTASBgNVHRMBAf8ECDAGAQH/AgEA
MA4GA1UdDwEB/wQEAwIBhjAdBgNVHSUEFjAUBggrBgEFBQcDAQYIKwYBBQUHAwIw
NAYIKwYBBQUHAQEEKDAmMCQGCCsGAQUFBzABhhhodHRwOi8vb2NzcC5kaWdpY2Vy
dC5jb20wSwYDVR0fBEQwQjBAoD6gPIY6aHR0cDovL2NybDQuZGlnaWNlcnQuY29t
L0RpZ2lDZXJ0SGlnaEFzc3VyYW5jZUVWUm9vdENBLmNybDA9BgNVHSAENjA0MDIG
BFUdIAAwKjAoBggrBgEFBQcCARYcaHR0cHM6Ly93d3cuZGlnaWNlcnQuY29tL0NQ
UzAdBgNVHQ4EFgQUUWj/kK8CB3U8zNllZGKiErhZcjswHwYDVR0jBBgwFoAUsT7D
aQP4v0cB1JgmGggC72NkK8MwDQYJKoZIhvcNAQELBQADggEBABiKlYkD5m3fXPwd
aOpKj4PWUS+Na0QWnqxj9dJubISZi6qBcYRb7TROsLd5kinMLYBq8I4g4Xmk/gNH
E+r1hspZcX30BJZr01lYPf7TMSVcGDiEo+afgv2MW5gxTs14nhr9hctJqvIni5ly
/D6q1UEL2tU2ob8cbkdJf17ZSHwD2f2LSaCYJkJA69aSEaRkCldUxPUd1gJea6zu
xICaEnL6VpPX/78whQYwvwt/Tv9XBZ0k7YXDK/umdaisLRbvfXknsuvCnQsH6qqF
0wGjIChBWUMo0oHjqvbsezt3tkBigAVBRQHvFwY+3sAzm2fTYS5yh+Rp/BIAV0Ae
cPUeybQ=
```

```
-----END CERTIFICATE-----
```

Step 2: Extract the public key (e, n) from the issuer's certificate.

To get the exponent we used penssl x509 -in c1.pem -text –noout.

```
                 c7:ed:6d:9b:f2:48:2f:f3:03:f5:a2
                 5e:e3
             Exponent: 65537 (0x10001)
     X509v3 extensions:
```

Then we also used the specified command to retrieve the mod for the n.

```
[11/08/21]seed@VM:~/.../lab7$ openssl x509 -in c1.pem -noout -modulus
Modulus=B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76073EC330558719E
94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9AED2C8B8433AD90B0BD5
D595F540AFC81DED4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B182842DC6ADA59CC71982A68
50F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233A30528C4BB28CC9AAC2B230D
78C67BE65E71B74A3E08FB81B71616A19D23124DE5D79208AC75A49CBACD17B21E4435657F532539D11
C0A9A631B199274680A37C2C25248CB395AA2B6E15DC1DDA020B821A293266F144A2141C7ED6D9BF248
2FF303F5A26892532F5EE3
```

Step 3: Extract the signature from the server's certificate.

We then proceeded to extract the signature from the server certificate using openssl x509 -in c0.pem -text –noout

```
Signature Algorithm: sha256WithRSAEncryption
     00:f3:bb:f2:3f:e1:d3:0f:c0:6e:10:cc:c1:47:66:68:10:16:
     59:dc:ff:1a:97:b5:a3:4b:a8:e3:48:cd:73:f3:9c:14:26:1d:
     08:b8:f3:5c:4a:80:04:78:8d:93:93:4e:49:e5:c0:e2:c1:5e:
     70:d7:bd:5e:ab:25:06:57:ba:dd:e9:c4:74:af:54:99:36:92:
     fb:b2:0c:ed:d1:0b:4b:ae:75:df:35:01:72:14:b1:de:8f:9e:
     3b:76:0f:a5:dd:ff:2a:54:02:83:24:c8:4f:bc:7a:e6:04:48:
     41:64:e0:79:67:ae:95:ed:37:b3:92:4c:65:58:65:09:34:68:
     9a:c3:20:db:25:5d:d9:94:2f:d1:3a:01:08:88:61:a4:48:a5:
     13:11:76:3e:2c:b4:6e:82:90:f2:69:7d:26:ae:59:ad:7d:91:
     17:99:ea:14:d0:47:97:fc:f4:be:b1:e7:4b:ac:ec:6b:96:96:
     61:fa:12:65:45:21:b8:5f:f4:43:b4:d9:00:37:09:c5:3b:6c:
     4d:62:2d:63:07:98:a7:14:eb:2b:61:9a:0b:2f:35:15:39:4e:
     29:31:bc:5e:fb:24:5b:fb:9f:5f:f2:f0:62:eb:a6:b9:8a:a4:
     1e:90:0d:fe:0f:03:c4:bd:44:e5:fd:47:38:30:7b:72:93:20:
     ce:aa:78:a5
```

Then modified it by deleting the spaces and the colons for it be usable.

```
[11/08/21]seed@VM:~/.../lab7$ vim signature
[11/08/21]seed@VM:~/.../lab7$ cat signature | tr -d '[:space:]:'
00f3bbf23fe1d30fc06e10ccc1476668101659dcff1a97b5a34ba8e348cd73f39c14261d08b8f35c4a8
004788d93934e49e5c0e2c15e70d7bd5eab250657badde9c474af54993692fbb20cedd10b4bae75df35
017214b1de8f9e3b760fa5ddff2a54028324c84fbc7ae604484164e07967ae95ed37b3924c655865093
4689ac320db255dd9942fd13a01088861a448a51311763e2cb46e8290f2697d26ae59ad7d911799ea14
d04797fcf4beb1e74bacec6b969661fa12654521b85ff443b4d9003709c53b6c4d622d630798a714eb2
b619a0b2f3515394e2931bc5efb245bfb9f5ff2f062eba6b98aa41e900dfe0f03c4bd44e5fd4738307b
729320ceaa78a5[11/08/21]seed@VM:~/.../lab7$
```

Step 4: Extract the body of the server's certificate.

4:d=1 hl=4 l=1560

```
[11/08/21]seed@VM:~/.../lab7$ openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=1840 cons: SEQUENCE
    4:d=1  hl=4 l=1560 cons:  SEQUENCE
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  16 prim:   INTEGER           :02493E07FA9E375A2DBBC61D94430FCF
```

1568:d1 hl=2 l=13

```
 1568:d=1  hl=2 l=  13 cons:  SEQUENCE
 1570:d=2  hl=2 l=   9 prim:   OBJECT            :sha256WithRSAEncryption
 1581:d=2  hl=2 l=   0 prim:   NULL
 1583:d=1  hl=4 l= 257 prim:  BIT STRING
```

The certificate body is from offset 4 to 1567 and the signature block is from 1568 to the end of the file.

```
[11/08/21]seed@VM:~/.../lab7$ openssl asn1parse -i -in c0.pem -strparse 4 -out c0_b
ody.bin -noout
[11/08/21]seed@VM:~/.../lab7$ sha256sum c0_body.bin
0640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38aefc2c17747d45f28  c0_body.bin
```

We used the -strparse option to get the field from the offset 4, to get the body of the certificate, while excluding the signature block.

Step 5: Verify the signature.

We first used python to create the message hash.

```
[11/08/21]seed@VM:~/.../lab7$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> prefix = "0001"
>>> hash = "0640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38aefc2c17747d45f28"
>>> A = "30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20".replace(' '.'')
  File "<stdin>", line 1
    A = "30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20".replace(' '.'')
                                                                                ^
SyntaxError: invalid syntax
>>> A = "30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20".replace(' ','')
>>> total_len = 256
>>> pad_len = total_len -1 - (len(A)+len(prefix)+len(hash))//2
>>> prefix + "AA" * pad_len + "00" + A +hash
'0001AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA003031
300d0609608648016503040201050004200640f8d13c0789ff0ed5437cf4bc9f2827d52146dddff38ae
fc2c17747d45f28'
```

We then reused the code we used in task 5, but now using the values we have extracted from the websites certificates.

```
        //modulos n value in a public key
        BN_hex2bn(&n, "B6E02FC22406C86D045FD7EF0A6406B27D22266516AE42409BCEDC9F9F76
073EC330558719B94F940E5A941F5556B4C2022AAFD098EE0B40D7C4D03B72C8149EEF90B111A9AED2C
8B8433AD90B0BD5D595F540AFC81DED4D9C5F57B786506899F58ADAD2C7051FA897C9DCA4B182842DC6
ADA59CC71982A6850F5E44582A378FFD35F10B0827325AF5BB8B9EA4BD51D027E2DD3B4233A30528C4B
B28CC9AAC2B230D78C67BE65E71B74A3E08FB81B71616A19D23124DE5D79208AC75A49CBACD17B21E44
35657F532539D11C0A9A631B199274680A37C2C25248CB395AA2B6E15DC1DDA020B821A293266F144A2
141C7ED6D9BF2482FF303F5A26892532F5EE3");
        //Exponent e value in a public key
        BN_dec2bn(&e, "65537");
        //message hash
        BN_hex2bn(&M, "0001AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA003031300d0609608648016503040201050004200640f8d13c0789ff0ed5437cf4b
c9f2827d52146dddff38aefc2c17747d45f28");
        //Signature
        BN_hex2bn(&S, "00f3bbf23fe1d30fc06e10ccc1476668101659dcff1a97b5a34ba8e348cd
73f39c14261d08b8f35c4a8004788d93934e49e5c0e2c15e70d7bd5eab250657badde9c474af5499369
2fbb20cedd10b4bae75df35017214b1de8f9e3b760fa5ddff2a54028324c84fbc7ae604484164e07967
ae95ed37b3924c6558650934689ac320db255dd9942fd13a01088861a448a51311763e2cb46e8290f26
97d26ae59ad7d911799ea14d04797fcf4beb1e74bacec6b969661fa12654521b85ff443b4d9003709c5
3b6c4d622d630798a714eb2b619a0b2f3515394e2931bc5efb245bfb9f5ff2f062eba6b98aa41e900df
e0f03c4bd44e5fd4738307b729320ceaa78a5");
```

Using BN_mod_exp(C,S,e,n,ctx) we got a valid signature and then checked again with the built in openssl verifying tool.

```
[11/08/21]seed@VM:~/.../lab7$ gcc -o task6 task7.c -lcrypto
[11/08/21]seed@VM:~/.../lab7$ ./task6
Valid
[11/08/21]seed@VM:~/.../lab7$ openssl verify -untrusted c1.pem c0.pem
c0.pem: OK
```